UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Desarrollo de un algoritmo genético para análisis de problemas basados en grafos

Diego Muñiz Velázquez
Tutor: Antonio González Pardo
Ponente: David Camacho Fernández

Junio 2018

Desarrollo de un algoritmo genético para análisis de problemas basados en grafos

AUTOR: Diego Muñiz Velázquez
TUTOR: Antonio González Pardo

Applied Intelligence & Data Analysis (AIDA)
Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2018

Resumen (castellano)

El uso de grafos para modelar diferentes problemas es un enfoque que nos puede ayudar a resolver un gran número de ellos. Aun con la ayuda de los grafos, muchos de estos problemas presentan una complejidad computacional elevada, lo que hace necesaria la implementación de algoritmos heurísticos para facilitar el tratamiento de la información que estos nos aportan.

Este Trabajo de Fin de Grado tratará el problema de la detección de comunidades en las Redes Sociales (RRSS). El imparable crecimiento de las RRSS y el inmenso número de usuarios conectados que en ellas se encuentran, las han convertido en un lugar donde conseguir una gran cantidad de información. Una de las características básicas de las RRSS, es la de permitir a sus usuarios agrupar, organizar y clasificar sus conexiones en diferentes grupos o "círculos". Los usuarios que formen un grupo tendrán una característica en común, pudiendo ser compañeros de trabajo, amigos o "hobbies" por ejemplo. Esta clasificación variará en función del usuario y la RS a la que estemos haciendo referencia. Al observar estas relaciones entre usuarios en las RRSS, rápidamente podemos observar que es posible tratarlas como un grafo donde, cada usuario se corresponde con un nodo y las relaciones entre usuarios con las aristas. En este TFG se pretende desarrollar un algoritmo heurístico bio-inspirado que determinará automáticamente los diferentes grupos de usuarios en un dataset específico (Facebook en nuestro caso).

Para evaluar la calidad de las soluciones encontradas se utilizará la densidad entre los nodos dentro y fuera de su comunidad, o grupo, para decidir si esta está bien definida o se podría mejorar.

Abstract (English)

Using of graphics to model different issues, is an approach that can help us solve many of them. Even with this help of graphs, many of those problems show a high computational complexity, which makes necessary implementing works of heuristics algorithms to ease the information treatment given by them.

This Final Degree Project will address the problem of detention of communities in Social Networks (SN) since due the unstoppable growth of many SN and the huge number of connected users in them, they have become a place to get a lot of information.
One of their basic features is to allow their users to group, organize and classify their connections in several groups or “circles”. Every user in a group will have a common feature each other, allowing them to be either co-workers, friends or members of a common hobby’s group, for example. This classify will change depending each user and SN. Analysing those relations between users in every SN, easily we’ll be able to understand that it is possible to treat them as a graph, when every user behaves as a node and their relations as edges. This FDP aims to develop a bio-inspired heuristic algorithm which will define automatically every different group of users in a specific dataset (Facebook in this case).

The quality of different obtained solutions will be evaluated by density between every node in and out its community or group, to decide if the quality is acceptable or it could be improved.

Palabras clave (castellano)

Redes Sociales, Algoritmo genético, Grafo, Círculos, Base de datos, Comunidades, Nodos, Individuos, Progenitores, Mutación, Cruce.

Keywords (inglés)

Social Networks, Genetic Algorithm, Graf, Círcles, Data Base, Comunities, Nodes, Singles, Parents, Mutation, Crossing.

Agradecimientos

Quisiera agradecer a mi tutor Antonio González Pardo por aguantar mis continuas preguntas y peticiones, así como el buen trato que ha tenido durante la tutela de este TFG.

De la misma manera agradecer a mi familia y pareja quienes de la misma forma han aguantado mis sollozos y me han dado animo en esta última fase de mi carrera.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Actualmente, las Redes Sociales (RRSS) son una parte fundamental de la mayoría de las personas en todo el mundo, habiendo cambiado su manera de actuar y relacionarse en muy poco tiempo. Con un crecimiento que no deja de aumentar día tras día y la cantidad de usuarios que conectan entre sí, parece claro que el correcto análisis y procesamiento de estas redes nos puede aportar una gran cantidad de información.

Dentro de las RRSS, la detección de comunidades se ha convertido en un campo de investigación de gran importancia cuya finalidad es la de encontrar y agrupar diferentes usuarios con características similares. Si conseguimos establecer estas comunidades y separar la red en varios grupos de usuarios con características parecidas, nos será mucho más sencillo predecir los gustos y necesidades, así como posibles comportamientos de un usuario basándonos en otros usuarios que pertenezcan a su misma comunidad.

Uno de los principales problemas encontrados al trabajar con las RRSS es la ingente cantidad de información que aportan, lo que supone un gran problema para los modelos de datos tradicionales que se usan habitualmente. Para poder abordar mejor este problema, se han utilizado los datos que nos aportan los usuarios de estas RRSS y se han representado en forma de inmensos grafos, como ya se hace en otros muchos problemas tales como el del coloreado de grafos. Para la detección de comunidades se usarán estos grafos donde los nodos representarán los usuarios y las aristas representarán las relaciones entre estos. Dichas relaciones serán diferentes en función del mecanismo que utilice cada red social, amistad (el cual será el caso que se tratará en este TFG con Facebook), seguimiento, trabajo…

Aunque desde un punto de vista social, siempre se ha tenido especial interés en estudiar la forma en la que las personas se relacionan con su entorno, crean diferentes amistades y el impacto que reciben de diferentes modas y tendencias. En este Trabajo de Fin de Grado nos centraremos en el punto de vista computacional, debido claro está, a la disciplina de estudios en la que se encuentra. Computacionalmente, la detección de comunidades supone un desafío en el análisis y procesamiento de grandes cantidades de datos, así como conseguir un tiempo de ejecución razonable de los algoritmos a implementar.

Para poder probar el funcionamiento de los algoritmos creados, se dispondrá de un dataset correspondiente a la red social Facebook presentada en forma de Ego Networks. Estas redes están formadas por un usuario central (el cual se quiere estudiar) que se denomina "ego", además esta red contiene aquellos usuarios (llamados alters) que están conectados con el ego, y todas las relaciones entre los alters. Además de contener la información de estas relaciones, también dispone de información de posibles comunidades encontradas entre los usuarios de esta red, permitiendo una vez programado el algoritmo deseado, comparar las comunidades obtenidas.

1.2 Objetivos

El objetivo principal de este Trabajo de Fin de Grado es el de crear un algoritmo genético en Python basado en el propuesto por [7] y aplicarlo al problema de detección de comunidades en las redes sociales. Más concretamente, aplicarlo sobre un grafo que representa una parte de la red social Facebook en forma de red ego.

Otro objetivo es, por tanto, el de construir un grafo a partir de estas redes ego, sobre el que poder trabajar y aplicar nuestro algoritmo.

Y finalmente, descubrir si la utilización de algoritmos genéticos aplicados sobre estas redes ego puede ayudar a la detección de comunidades en las redes sociales, y siendo así, ayudar de la misma forma a entender su funcionamiento, formación, y comprender aún mejor como se propaga y distribuye la información a través de ellas.

Consiguiendo estos objetivos, ayudaremos a su vez a comprender la forma en la que la sociedad se relaciona y se ve afectada por tendencias y modas en el mundo real.

1.3 Organización de la memoria

Esta memoria consta de los siguientes capítulos:

- **Apartado 2:** Estado del arte. Donde se presentarán los desarrollos actuales en los campos que se tratan en este Trabajo de Fin de Grado, se detallarán otros trabajos similares y diferentes aproximaciones en la detección de comunidades, así como las ventajas e inconvenientes que estas suponen.

- **Apartado 3:** Diseño. En este apartado se explicará cómo se ha diseñado el algoritmo genético presentado y algunos conceptos a tener en cuenta para entender correctamente el funcionamiento del mismo.
- **Apartado 4:** Desarrollo. Aquí se detallará la implementación del algoritmo genético diseñado, así como el funcionamiento de sus diferentes funciones y problemas encontrados durante el proceso de desarrollo.
- **Apartado 5:** Pruebas y resultados. Esta sección mostrará los resultados obtenidos de la aplicación del algoritmo genético desarrollado sobre los dataset que disponemos, mostrando además diferentes tablas y gráficos de los resultados obtenidos y su comparación con las comunidades que nos proporciona el propio dataset.
- **Apartado 6:** Conclusiones y trabajo futuro. Reflexión de los resultados obtenidos del algoritmo genético desarrollado, y en función de los mismos, comentar que se podría mejorar o arreglar en trabajos futuros.

Al final de la memoria se encuentran las referencias usadas para el desarrollo de este Trabajo de Fin de Grado.

2 Estado del arte

Este trabajo de Fin de Grado trata sobre el uso de algoritmos genéticos para la búsqueda de comunidades en las redes sociales representadas mediante grafos, por lo que en este apartado se explicarán cada uno de estos componentes, además de localizarlos histórica y tecnológicamente.

2.1 Algoritmos Genéticos

Los Algoritmos Genéticos son métodos adaptativos cuya finalidad es la de resolver problemas de búsqueda y adaptación. Como su propio nombre indica, estos métodos están basados en el proceso genético de los organismos vivos. Este proceso se lleva a cabo durante el paso de varias generaciones, donde las poblaciones van evolucionando acorde con los principios de la selección natural y la supervivencia del más fuerte [1] (Darwin, 1859). Los Algoritmos Genéticos intentarán imitar este proceso de selección, evolucionando y creando soluciones óptimas para diversos problemas del mundo real.

En la naturaleza, la selección entre los individuos de una población se realiza mediante la continua competencia entre sí por la búsqueda y obtención de recursos, siendo solo los individuos con más éxito a en esta lucha, los que serán capaces de atraer a más individuos y de generar una mayor descendencia. Por el contrario, los que no se adapten en esta carrera por la supervivencia, verán un decremento en el número de descendientes. De esta manera, serán los genes de los individuos mejor adaptados los que más se propaguen, y con ellos, la creación de una descendencia más fuerte y mejor adaptada al entorno, permitiendo que el número de individuos siga una tendencia creciente.

De forma análoga, los Algoritmos Genéticos utilizarán el mismo comportamiento natural. Se componen de una población de individuos, cada uno de los cuales, porta con una solución posible al problema que se enfrenta. Para determinar la capacidad que tiene cada individuo de sobrevivir, se les asignará un valor indicando cuan buena es la solución ofrecida, lo que equivaldría a la capacidad en la naturaleza de un individuo a la hora de obtener recursos y adaptarse. Cuanto mejor sea este valor, más probabilidades tendrá el individuo de reproducirse, cruzando la información que porta con otro individuo seleccionado de la misma forma. Estos cruces generarán una descendencia, la cual tendrá características de cada uno de sus progenitores, propagando las buenas características que

los hacían mejores al resto y, con suerte, mejorando las soluciones al problema que se estaban obteniendo.

Los principios básicos de los Algoritmos Genéticos fueron establecidos por [3] (Hollan, 1975) y se encuentran descritos en varios textos - [2] (Goldberg, 1989), [4] (Davis, 1991), [5] (Michalwiez, 1992), [6] (Reeves, 1993) -, en los cuales no vamos a profundizar puesto que no es el tema de estudio al que este Trabajo de Fin de Grado está orientado.

Aunque los Algoritmos Genéticos generalmente han sido asociados a funciones de optimización, el rango de problemas a los cuales han sido y pueden ser aplicados es bastante amplio. Algunos ejemplos del tipo de estos problemas son, de decisión y estrategia, diseño y parametrización, planificación y asignación de recursos y predicción. De la misma forma, los campos en los que se han aplicado estos algoritmos son igual de amplios, desde la Acústica y la Ingeniería aeroespacial [8, 9] para la optimización de salas acústicas y diseños de alas en los aviones, como para métodos de predicción en el campo de los Mercados financieros [10].

En cuanto a la composición de los Algoritmos Genéticos, existen tres operadores fundamentales que le permiten realizar los procesos de evolución y selección anteriormente descritos [11].

- **Cruce:** operador que intenta emular la creación de descendencia entre los individuos de la población. Existen varias formas de cruzar los individuos, intercambiando todos los genes a partir de cierto punto, en varios puntos, o mediante máscaras.
- **Mutación:** en esta parte, el algoritmo imita las mutaciones en los genes que se dan en la naturaleza, produciéndose cambios en el color de los ojos, pelo o que en algunos casos pueden llevar a el padecimiento de enfermedades. Por lo general, la mutación de los individuos que componen un Algoritmo Genético, se realizará de manera puntual y en pocos genes, produciéndose cambios de orden, inversiones o directamente produciéndose un cambio de unos genes por otros.
- **Aceptación:** en este proceso, se emula la adaptación del individuo con su entorno. Los individuos, ya con su índice de supervivencia calculado, competirán por ver cuáles de ellos sobreviven y pasan a la siguiente generación del algoritmo. Dependiendo del algoritmo, las formas de aceptación pueden variar, llegando

incluso en algunos casos a aceptarse todos los individuos hasta la finalización de la ejecución.

Mas adelante, en el apartado de diseño, se profundizará más en cómo funcionan las diferentes operaciones y del flujo que los individuos realizan en cada generación.

2.2 Detección de comunidades.

Como se ha introducido en el apartado anterior, los datos que componen las redes sociales pueden expresarse mediante teoría de grafos como un grafo compuesto por nodos y aristas. La detección de comunidades en las redes sociales tiene como finalidad, encontrar subconjuntos más pequeños donde las similitudes entre los nodos que los componen se maximicen, y por el contrario, minimizar estas similitudes con los nodos de otros subconjuntos.

La detección de comunidades ha pasado a ser uno de los problemas que, con más interés, se estudia a la hora de analizar las redes sociales, ya que nos aporta gran cantidad de información sobre el comportamiento actual de la sociedad. Este problema se basa en descubrir cómo se organizan las diferentes redes, o lo que es lo mismo, en la detección de subconjuntos de redes dentro de la red principal. Si representamos estas redes como grafos, donde los usuarios se representen mediante nodos y sus conexiones mediante los enlaces en la red, usando teoría de grafos, podremos analizar las características de cada nodo e identificar que rasgos son comunes entre nodos y que rasgos los diferencian, obteniendo de esta forma, los subconjuntos de grafos que nos interesan.

Los problemas de grafos no solo se han utilizado para este tipo de problemas. Históricamente el primer estudio de un grafo fue trabajo de Leonhard Euler, en 1736, sobre el problema de los puentes de Königsberg [13], el cual se considera el primer resultado de la teoría de grafos. Desde entonces la construcción de grafos ha servido para resolver diversos problemas de diferentes características, el más importante posiblemente sea el que se planteó con el problema de los cuatro colores, el cual afirma que es posible, usando solo cuatro colores, pintar cualquier mapa sin que dos países adyacentes compartan color.

En el mundo de la computación, que es donde está ubicado este Trabajo de Fin de Grado, los problemas de grafos siempre han sido de utilidad para expresar gran cantidad de

problemas computacionales, como por ejemplo, máquinas de estados, árboles de decisión, algoritmos de distancias… o para llevar a cabo decisiones basadas en el aprendizaje automático [11] e incluso tareas orientadas al hardware, permitiéndonos analizar la infraestructura necesaria para el desarrollo de la web (enrutamiento, enlaces, distancias entre repetidores…). Por todos estos motivos, convertir nuestro problema en un problema de grafos nos será de gran utilidad, ya que, al ser un campo ampliamente estudiado en el ámbito de la informática, dispondremos de una gran diversidad de herramientas y documentación que nos facilitará el análisis de las redes sociales y ayudará en la implementación del algoritmo deseado.

Teniendo claros los conceptos sobre los que vamos a trabajar y su recorrido a lo largo de la historia, de la importancia del problema que queremos resolver sobre el análisis de las redes sociales y de las herramientas que vamos a usar para ello, intentaremos abordar y resolver dicho problema dando respuesta a las preguntas realizadas anteriormente en la introducción de este Trabajo de Fin de Grado.

3 Diseño

3.1 Representación de los datos en un grafo.

Para poder operar con la gran cantidad de datos que las redes sociales nos aportan, es importante encontrar una buena forma de representar los usuarios y relaciones que tienen entre sí.

Como se ha mencionado durante la introducción, la forma elegida de representar estos datos será mediante las ”Ego Networks”. Usamos este tipo de redes, porque como ya hemos comentado, el tamaño de las redes sociales es muy grande, y el único modo de poder operar con tanta cantidad de datos es bien usando técnicas de big data, o tratar de reducir el tamaño de los datos. En este trabajo se ha decidido optar por lo segundo, no solo por simplicidad, sino también porque las redes ego nos aportan las relaciones que tiene un usuario en concreto (ego), que es exactamente el tema de estudio que estamos buscando.

Como el objetivo en este Trabajo de Fin de Grado es el de aplicar un algoritmo genético sobre un problema de grafos para la detección de comunidades, se crearán grafos que representen estas redes ego, de tal forma que cada usuario se mostrará mediante un nodo y las relaciones descritas en las redes se mostrarán mediante las conexiones entre ellos.

Una vez tengamos los datos de una red social en forma de estas redes y correctamente representadas mediante grafos, podremos aplicar el algoritmo deseado y empezar con la búsqueda de las comunidades dentro de la red.

3.2 Obtención de los datos de redes sociales.

Los datos que usaremos para comprobar el funcionamiento de nuestro algoritmo los obtendremos de [12], donde podemos obtener los datos de varios usuarios de distintas redes sociales, en nuestro caso elegiremos Facebook. Estos datasets consisten en ‘círculos’ (o ‘listas de amigos’) de esta red social, obteniéndose de varios usuarios que usan su app para conectarse. Estos datasets están formados por las características de cada nodo (perfiles de usuario), los círculos y las redes ego.

Estos datos sobre la red social Facebook, han sido anonimizados reemplazando los valores de las características por otro valor, si originalmente los datos nos indican

"political=Democratic Party", el dato anonimizado será "political=anonymized feature 1". Aunque veremos que las características de cada nodo no nos resultarán relevantes para la implementación de nuestro algoritmo, es importante dejar claro que tanto los nombres de usuario como las características de cada uno están anonimizadas para mantener su privacidad.

A continuación, se detallará el contenido de cada fichero correspondiente a cada una de las redes ego de las que dispone este dataset.

- **Circles:** En este fichero se especifican los círculos que se encuentran en cada red ego y los nodos que los componen. Utilizaremos este fichero para comparar los resultados que se obtengan al aplicar el algoritmo genético sobre los datos de la red ego, dándonos una idea de si los resultados obtenidos son coherentes o distan mucho de los círculos aquí representados.
- **Edges:** Fichero donde se especifican las conexiones entre nodos, o lo que es lo mismo, que los usuarios correspondientes a esos nodos están conectados en esa red social. En el caso de Facebook estas relaciones son bidireccionales (ambos usuarios están conectados entre sí) y en el caso de Twitter serían unidireccionales (puesto que seguir a un usuario no implica que este también te siga a ti).
- **EgoFeat y FeatNames:** En estos ficheros, es donde se encontrarán los datos de los perfiles de usuario pertenecientes al ego de cada red y a qué dato le corresponde cada valor. Estos datos, como explicamos anteriormente, están anonimizados. Como para el funcionamiento de nuestro algoritmo solo nos interesan las relaciones entre usuarios, estos ficheros no nos aportarán ningún dato relevante para su funcionamiento.
- **Feat:** En estos ficheros, es donde se encontrarán los datos de los perfiles de cada usuario relacionado con nuestro ego dentro de cada red. De este fichero solo nos importará el primer dato de cada fila, ya que es el que nos indica el nombre que se le da a cada nodo, para posteriormente, poder identificar las conexiones entre ellos en el fichero “edges” y saber cuántos nodos va a tener nuestro grafo.

3.3 Algoritmo genético.

Para cumplir nuestro objetivo en la búsqueda de comunidades dentro de las redes sociales, se implementará un algoritmo genético con el que poder obtener varias posibles soluciones

e ir mejorándolas con el paso de varias generaciones, hasta finalmente, obtener una solución que consideremos suficientemente buena para la red en la que se ha aplicado.

Se ha decidido usar un algoritmo genético debido a que, a pesar de usar redes ego, la complejidad de los algoritmos de detección de comunidades es muy alta, obligándonos a recurrir al uso de algoritmos heurísticos, y dentro de los algoritmos heurísticos, los genéticos son de los más conocidos.

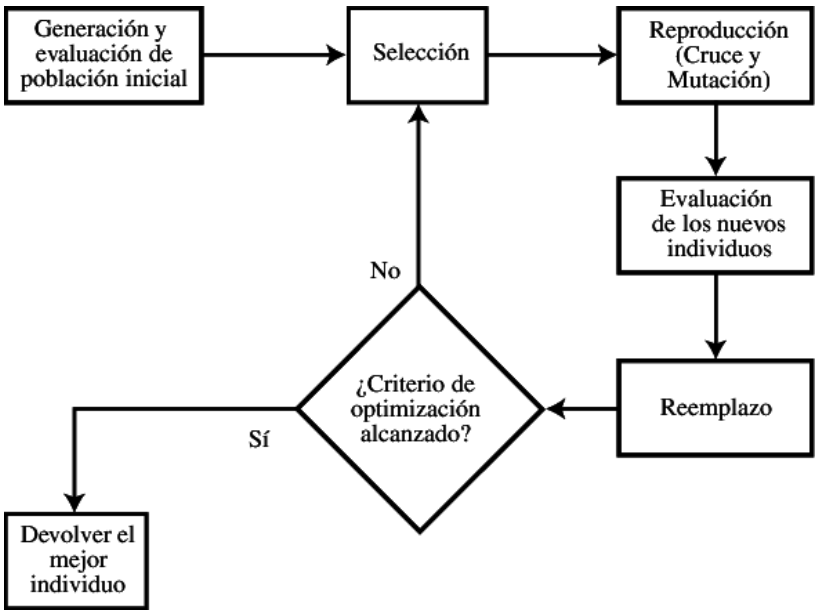

Figura 3-1: Diagrama de flujo algoritmo genético

Utilizaremos *Figura 3-1* como ejemplo visual para explicar cómo funcionan cada una de las fases de un algoritmo genético básico.

- **Generación de la población inicial:** Esta es la fase inicial de cualquier algoritmo genético, en ella se inicializarán todos los individuos con los que el algoritmo genético empezará a evaluar y modificar. Cada individuo se suele representar como una lista de datos que representa una posible solución a un problema dado. Esta generación se lleva a cabo de diferentes maneras en función de los criterios de cada algoritmo, pudiendo hacerse completamente aleatoria o bien mediante unos criterios específicos. El número de individuos generado suele ser variable, pudiendo así modificarlo para poder realizar pruebas con distintas cantidades de individuos.

- **Evaluación de la población:** En esta fase realizaremos las operaciones necesarias para determinar la validez de cada individuo, lo que generalmente se suele llamar fitness. La evaluación del fitness cambiará en función del problema al que se esté enfrentando cada algoritmo genético, ya sea contar la cantidad de datos diferentes o la suma de ellos, en cualquier caso, el fitness obtenido evaluará cómo de buena es la solución que nos está aportando cada uno.
- **Selección:** Como su propio nombre indica, en esta fase se seleccionarán los individuos con mejores resultados según el fitness propuesto y se descartarán un porcentaje de los peores. Esta fase en algunos algoritmos puede no aplicarse y en su lugar continuar con todos los individuos de los que se dispone.
- **Reproducción:** Durante esta fase los individuos se "reproducirán" generando nuevos individuos en base a los datos de sus progenitores. El proceso de cruce es el encargado de juntar los datos de dos individuos padre con los que se generarán uno o más hijos.

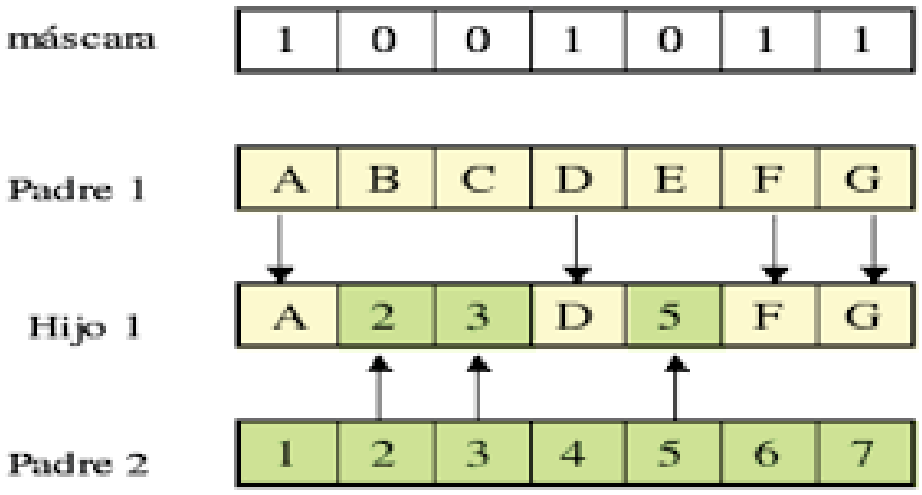

Figura 3-2: Diagrama de cruce

Existen diferentes tipos de cruce con los que obtener nueva descendencia y cualquiera de ellos es válido.

La mutación se llevará a cabo sobre la población resultante tras realizar las operaciones de cruce, donde el valor de los datos de cada individuo podrá verse modificado por otro. Este proceso intenta imitar las mutaciones en los genes cuando se genera una descendencia entre dos individuos, produciéndose una

variación en el resultado que debería de obtenerse. Tanto la selección de que dato muta en cada individuo como el valor que puede tomar, suele ser aleatorio, aunque puede darse el caso en el que se quiera especificar los valores a los que se puede mutar.

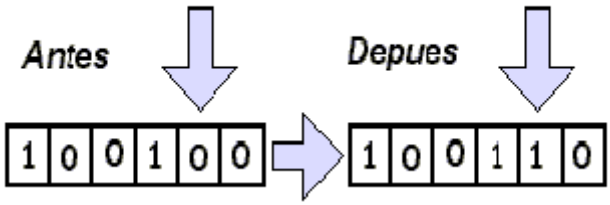

Figura 3-3: Diagrama de mutación

- **Reemplazo:** En este paso simplemente cambiamos los nuevos individuos creados con los que ya teníamos.
- **Criterio de optimización:** A esta fase la podemos denominar como la condición de parada de nuestro algoritmo, puesto que se comprobará si las soluciones obtenidas tras la evaluación de los individuos, es suficientemente buena como para parar la ejecución y devolver los resultados obtenidos o, sí por el contrario, se desea continuar con la ejecución del algoritmo genético para seguir obteniendo nuevas soluciones. También se suele poner un tope al número de iteraciones que puede dar el algoritmo, ya que, si no se llega a un criterio de optimización conforme, el algoritmo se detenga y devuelva la mejor solución encontrada.

A continuación, se detallará el diseño de las fases que contendrá nuestro algoritmo.

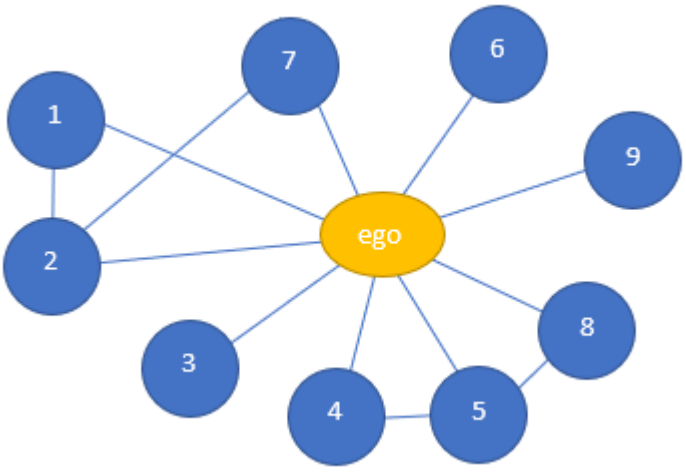

Figura 3-4: Grafo de prueba

3.3.1 Población inicial.

Al inicializar nuestro grafo, guardaremos en una lista el nombre de todos los nodos que lo componen, excluyendo el nodo ego, ya que no nos será necesario operar con él. Esto nos permitirá referirnos a ellos mediante su índice de posición en esa lista y no por los nombres que puedan tener.

La forma de nuestros individuos consistirá en listas de un tamaño igual al número total de nodos que tiene nuestro grafo menos el ego. Estas listas estarán formadas por los nombres de los nodos que hay en nuestro grafo, y nos indicarán que relaciones tienen entre ellos.

La forma con la que indicaremos estas relaciones será de la siguiente manera:
Como los individuos son listas del mismo tamaño a la lista total de nodos, cuando queramos inicializar la relación correspondiente al nodo "1", solo tendremos que ver qué posición ocupa en nuestra lista total de nodos y modificar la misma posición en el individuo con el valor que le corresponda. Suponiendo el grafo *Figura 3-4* la lista con el nombre de nuestros nodos podría ser [1,2,3,4,5,6,7,8,9], por lo que si queremos indicar las relaciones del nodo con nombre "1" tendremos que modificar la primera posición en las listas de los individuos, ya que "1" se encuentra en la primera posición de la lista, si queremos modificar las del "2" tendremos que modificar la segunda posición etc.

El dato que guardaremos en cada posición de nuestro individuo será el nombre de un nodo vecino al nodo del que estamos haciendo referencia, esta elección se hará de forma aleatoria al inicializar todos los individuos, eligiendo un vecino de todos los posibles.
Si volvemos a utilizar el grafo de prueba *Figura 3-4*, podemos ver como los vecinos del nodo "2" son el "1" y el "7" los del "5" el "4" y el "8"… por lo que al inicializar la segunda posición de uno de nuestros individuos, el cual hace referencia al nodo "2", podrá tomar valores "1" o "7", ya que son los 2 posibles vecinos de los que dispone, al inicializarse la quinta posición, esta tomará el valor "4" u "8", que son los vecinos del nodo "5" etc. Si se diera el caso en el que un nodo no dispone de posibles vecinos, el valor que tomará en la posición correspondiente de cada individuo será la de su propio nombre, como es el caso de la tercera posición de cada individuo, el cual solo podrá optar por el valor "3".

De esta forma tendremos una población inicial de individuos con los que empezar a operar y buscar las posibles comunidades de la red.

El algoritmo implementado deberá generar un número de individuos igual al que le indiquemos cuando se inicialice.

3.3.2 Tratamiento de los individuos.

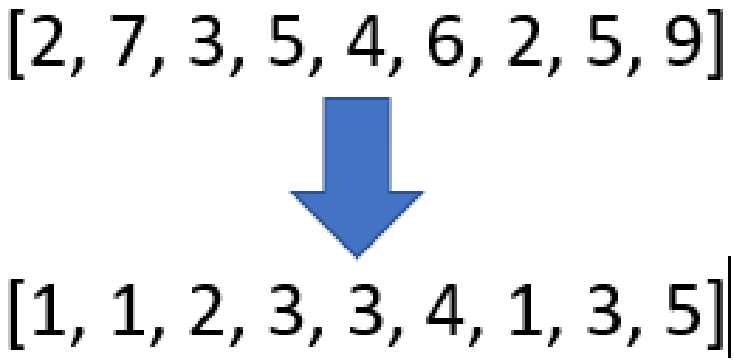

Figura 3-5: Transformación de un individuo a índices de comunidad

Para que la función de fitness que hemos diseñado pueda realizar las operaciones correspondientes en cada generación, vamos a necesitar transformar los datos que contiene cada individuo por otros que nos indiquen "comunidades". Sustituiremos los datos que hacen referencia al nombre de los nodos por el del número de comunidad al que creemos que pertenece, empezando en *1* hasta *n* comunidades diferentes.

Siguiendo con el ejemplo del grafo anterior, un ejemplo de individuo posible con su transformación a comunidades sería la mostrada en *Figura 3-5*, donde nos indicaría que los nodos “1”, “2” y “7” formarían la comunidad 1, el “4”, “5” y “8” la comunidad 2 etc.

3.3.3 Cruce de individuos.

La función de cruce de nuestro algoritmo genético consistirá en el intercambio de valores entre los nodos de dos individuos, obteniendo de cada cruce un nuevo individuo mezcla de sus “padres”.

La selección de los individuos para el cruce se realiza mediante un método llamado “torneo”. Este método consistirá en la elección al azar de un número *k* entre 2 y el número total de individuos, el valor que tome *k* será la cantidad de individuos que escogeremos al azar entre toda nuestra población, y entre todos los elementos seleccionados, se escogerán para el cruce los dos con mejor fitness. De esta manera intentamos dar la posibilidad de que cualquier individuo pueda cruzarse, pero sin dejar de seleccionar a los mejores para poder conseguir un mejor resultado cada vez.

A la hora de escoger con qué valor quedarnos al cruzar los progenitores, se generará una lista de 1’s y 0’s aleatorios del mismo tamaño al de los individuos, la cual llamaremos máscara. Esta máscara nos servirá para indicarnos si el valor a elegir es el del primer o del segundo progenitor, siendo por ejemplo un 0 el primer progenitor y un 1 el segundo.

Los valores a cruzar serán los nombres de los nodos y no los índices de las comunidades, ya que estos índices solo los utilizaremos a la hora de realizar las operaciones en la función fitness.

La forma en la que este cruce se va a llevar a cabo está representada en la figura *Figura 3-2.*

3.3.4 Mutación de individuos.

A la hora de realizar la mutación en nuestros individuos, haremos que cualquier valor de ellos se pueda cambiar aleatoriamente por el de cualquiera de sus vecinos, de la misma forma en la que seleccionamos los valores a la hora de inicializar la población. La selección de en qué posición se realizará la mutación se hará también de forma aleatoria,

indicando al comienzo de la ejecución del algoritmo con qué probabilidad queremos que esto suceda, permitiéndonos ajustar con facilidad la frecuencia de este hecho y realizar pruebas con diferentes variaciones. El número de datos que puede mutar en cada individuo será igual al del tamaño de este, siendo la probabilidad que indiquemos la única limitación para ello. Si a la hora de realizar la mutación en una posición donde el nodo al que se hace referencia no tiene ningún vecino, se realizará el mismo procedimiento que en la inicialización, dejando el mismo valor que ya estaba colocado.

3.3.5 Fitness.

Para comprobar si nuestro individuo tiene unas comunidades bien definidas, se le medirán dos características, densidad "intra-cluster" y densidad "inter-cluster".

Al realizar la detección de comunidades es importante buscar y detectar regiones en la red donde se maximicen las similitudes entre los elementos que la componen y minimizar sus diferencias, de la misma forma en la que buscamos que las diferencias entre los usuarios que componen dicha región con los que no están, sean mayores.

La densidad "intra-cluster" se puede representar de la siguiente manera:

$$\delta_{int}(\mathcal{C}) = \frac{\#\text{ internal edges of } \mathcal{C}}{n_c(n_c - 1)/2}.$$

Figura 3-6: Densidad intra-cluster

Definida como el número de aristas internas de la comunidad (internal edges of C) entre la mitad del producto del número de nodos contenidos en la comunidad por el número de nodos contenidos en la comunidad menos uno, representando el total de aristas posibles dentro de la comunidad.

La densidad "inter-cluster" se puede representar de la siguiente manera:

$$\delta_{ext}(\mathcal{C}) = \frac{\#\text{ inter-cluster edges of } \mathcal{C}}{n_c(n - n_c)}.$$

Figura 3-7: Densidad inter-cluster

Definida como la división del número de conexiones entre nodos contenidos en la comunidad y fuera de ella (inter-cluster edges of C) entre el producto del número de nodos de la comunidad por la diferencia entre el número de nodos del grafo en la comunidad, representando el número de conexiones intercomunitarias presentes entre las posibles.

Como lo que queremos es que la densidad intra-cluster sea lo más grande posible, y que la densidad inter-cluster sea la menor posible, a la hora de decidir si las comunidades encontradas son óptimas, intentaremos que la diferencia de la densidad intra-cluster con la densidad inter-cluster sea lo mayor posible, puesto queremos que los nodos que relacionamos en la misma comunidad tengan una gran cohesión, frente a los que pertenecen a otras comunidades, donde queremos que tengan la menor cohesión posible.

Tomaremos cada individuo de la población y comprobaremos las densidades de sus comunidades calculando así el valor del fitness de cada uno.
El objetivo será encontrar un individuo cuyas comunidades tengan el fitness más alto de toda la población.

3.3.6 Elitismo.

Como se ha comentado a la hora de explicar las diferentes fases de las que se compone un algoritmo genético, algunos seleccionan los individuos con mejor fitness y desechan un porcentaje de los peores para evitar que evolucionen y creen peores individuos.

En nuestro caso, no vamos a descartar ese porcentaje de individuos con peor fitness, sino que vamos a seleccionar un porcentaje de los mejores individuos y los añadiremos después de que toda la población haya realizado los procesos de evolución, consiguiendo que los

mejores individuos de cada iteración no se pierdan por un mal cruce o una mutación desafortunada.

Llamaremos a esta característica elitismo, haciendo referencia a la élite de la población.

Como con el resto de variables en nuestro algoritmo, el porcentaje de individuos respecto al total que pertenecerá al grupo elitista se indicará e inicializará por parámetro al comienzo de la ejecución, permitiéndonos cambiar fácilmente su valor y realizar varias pruebas con diferentes valores, para entender cómo influye dicho parámetro en el rendimiento del algoritmo.

3.3.7 Condición de Parada.

Finalmente, necesitaremos determinar en qué momento nuestro algoritmo encuentra una solución suficientemente buena como terminar su ejecución y devolver el mejor resultado encontrado, y este momento será cuando calculemos que la variación del fitness medio de la actual población con la anterior no supere un delta indicado.

Durante las ejecuciones del algoritmo se irán calculando y almacenando el fitness medio de los individuos que componen cada población, si en algún momento se detecta que la mejora o deterioro de este valor no es suficientemente grande, determinaremos que hemos llegado a una zona donde nuestro algoritmo genético ya no es capaz de encontrar mejores soluciones y deberá terminar devolviendo la mejor solución encontrada hasta el momento.

También se añadirá un tope al número de generaciones que nuestro algoritmo puede llegar a producir, para evitar que el delta especificado no llegue a cumplirse o se cumpla demasiado tarde.

Tanto el delta como el número máximo de generaciones son variables que se pasarán por parámetro de la misma forma que las anteriores variables nombradas.

3.3.8 Valor de retorno.

Una vez determinados todos los valores de las variables y ejecutado nuestro algoritmo genético, este tendrá que aportarnos una solución con la que poder identificar las

comunidades encontradas en el grafo donde se ha aplicado, así como el fitness que esas comunidades han conseguido. Con el fin de responder a las preguntas que se han marcado en este Trabajo de Fin de Grado, este algoritmo genético mostrará al finalizar, una lista de listas donde se agruparán los nombres de los nodos que consideremos se encuentran en la misma comunidad, permitiéndonos compararlas con los grafos creados y con otras posibles soluciones que aporten otro tipo de algoritmos.

También se mostrará el valor del fitness medio de la población en cada generación, con el fin de obtener un análisis de la evolución que sufren los individuos con diferentes valores de mutación, elitismo, generaciones, y número máximo de individuos.

Con todos estos datos, intentaremos descubrir si es posible aplicar un algoritmo genético sobre un problema de grafos para el descubrimiento de comunidades en las redes sociales y de ser así, cuan buena es esta solución, posibles mejoras etc.

4 Desarrollo

4.1 Creación del grafo

Lo primero que deberemos hacer, será leer los datos de una red ego correspondiente a la red social de la que queramos obtener información, con el fin de detectar las posibles comunidades que pueda haber. Para ello, usaremos los datos que nos aporta el ya mencionado dataset [12].

Para poder identificar los usuarios, y, por tanto, los nodos que van a componer nuestro grafo, empezaremos leyendo el fichero "feat.*x*", que como ya explicamos anteriormente, contiene en la primera columna el nombre identificativo de cada usuario. Durante este proceso, guardaremos en una lista los nombres leídos, para posteriormente, poder pasársela a la función que nos permitirá crear el grafo asociado.

La lectura del fichero "edges.*x*", nos facilitará obtener las relaciones, que los nodos anteriormente guardados, tienen entre sí. Para la correcta lectura de las conexiones cuando llamemos a la función generadora del grafo, estas se deberán guardar como una lista de tuplas, donde en cada una, colocaremos el nombre de los nodos que tienen una relación. Siendo un ejemplo posible: [("1","2"), ("2","7")], donde nos indicaría que los nodos "1" y "2" estarán relacionados en nuestro grafo, al igual que los nodos "2" y "7".

Para la construcción del grafo con los datos que hemos obtenido, usaremos la librería de Python Networkx [14].

Networkx nos permitirá construir un grafo que represente la red ego de la que hemos obtenido los datos. Primero se le pasará al constructor del grafo los nombres de los nodos, a los que se les añadirá un nodo adicional representando al ego, inicializando todos los nodos que van a componer nuestro grafo. Posteriormente se le pasarán al constructor todos los enlaces leídos del fichero "edges.*x*", construyendo todas las conexiones entre los nodos que teníamos creados. Adicionalmente, y como es lógico, añadiremos un enlace por cada nodo con el nodo ego. Una vez completadas estas acciones, obtendremos la representación en forma de grafo de una red ego, donde todos los nodos tendrán una conexión con el nodo ego, y podrán tener o no, conexiones con el resto de los nodos.

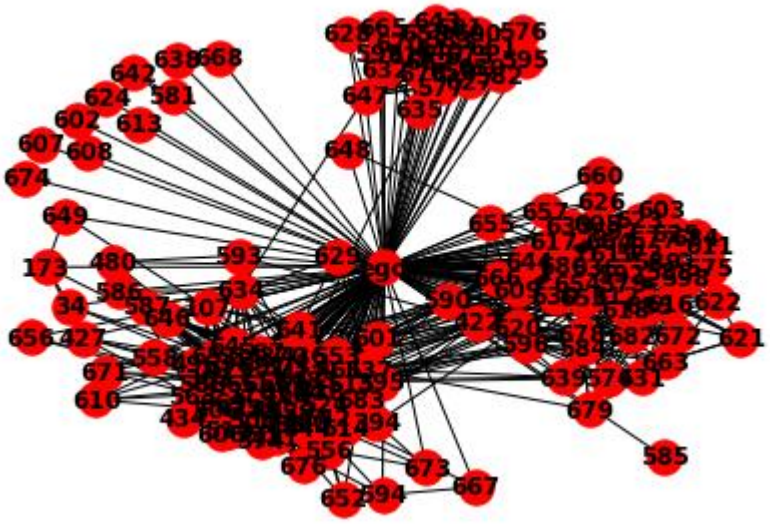

Figura 4-1: Ejemplo de grafo generado con Networkx sobre red ego

4.2 Algoritmo genético.

Una vez creado el grafo correspondiente a la red, los datos se pasarán al algoritmo genético que se ha implementado para detectar las posibles comunidades que en esta red se encuentran.

El algoritmo consta de un constructor, donde se inicializan todos los parámetros variables de las operaciones, así como el grafo creado anteriormente. Estos datos serán accesibles en el resto de las funciones de las que consta el algoritmo.

Para iniciar su funcionamiento, se llamará a la función principal del algoritmo donde se realizarán las diferentes operaciones descritas en el apartado de diseño. Se empezará inicializando los individuos en base al grafo construido, y comenzará la iteración de las fases de evaluación, cruce, mutación y selección, hasta encontrar una detección de comunidades que consideremos óptima.

Mientras se estén calculando los fitness de los individuos, se irán comparando al mismo tiempo con el mejor valor obtenido hasta el momento, y se actualizará en el caso de que el nuevo valor sea mejor que el guardado. De la misma manera, en cada iteración, se irá actualizando el fitness medio obtenido por la población, para, llegado el momento de comprobar el delta establecido en la condición de parada, poder comparar la media de la

población anterior. Si el valor absoluto de la diferencia entre ambas medias es menor al delta establecido, la ejecución del algoritmo terminará, devolviendo el mejor conjunto de comunidades obtenido.

Adicionalmente, se irá imprimiendo por pantalla en cada iteración, el fitness medio de la población actual, con el fin de identificar que el algoritmo está funcionando y de llevar un registro en la evolución de los individuos.

Por último, cabe recordar, que este algoritmo dará también por terminado su análisis en el caso de que se cumpla el número máximo de generaciones indicado en la inicialización.

4.2.1 Inicialización.

El proceso de inicialización de los primeros individuos de nuestra población se llevará a cabo en una función propia.

Esta función devolverá una lista con un número de listas en su interior igual al número de la población que se haya indicado en la llamada al constructor del algoritmo. Las listas en el interior contendrán los nombres de los nodos de la forma que se explicó en el punto 3.3.1.

A la hora de inicializar los individuos, iremos índice por índice accediendo a cada una de las posiciones de sus listas, y comprobaremos el nombre del nodo al que estamos haciendo referencia accediendo, en la misma posición, a la lista donde guardamos todos los nodos de nuestro grafo. Una vez sepamos a que nodo le estamos asignando el valor, utilizaremos la función que Networkx nos ofrece para encontrar los vecinos a un nodo dado. La devolución de esta función será una lista de vecinos del nodo que le hemos pasado por parámetro. Para poder seleccionar uno de estos vecinos aleatoriamente, tal y como se detalló en la fase de diseño, se hará uso de la librería random de Python, la cual nos ofrece la posibilidad de, dada una lista de elementos, devolvernos uno de ellos de forma aleatoria. El elemento seleccionado se asignará a la posición que estábamos inicializando y pasaremos a la siguiente, hasta terminar de inicializar todos los índices de todos los individuos que componen nuestra población.

Al finalizar la inicialización de todos los individuos, nuestro algoritmo ya dispondrá de una población preparada para empezar a evolucionar obtener los resultados.

4.2.2 Normalización.

Para que nuestro algoritmo de fitness pueda operar con los datos que hemos inicializado, deberemos transformarlo de la forma en la que indicamos en el punto 3.3.2, con los que poder obtener los índices de las comunidades a las que pertenece cada nodo.

Este proceso se ha decidido realizar en dos pasos. Primero se cambiarán los nombres de los nodos en las listas de los individuos por el correspondiente índice en nuestra lista total de nodos pertenecientes al grafo, esto ayudará a nuestra función de normalización a poder relacionar los individuos con sus comunidades más fácilmente. Una vez cambiados los datos de los individuos, se llamará a la función encargada de encontrar las comunidades. Esta función inicializará una lista de -1's del mismo tamaño que las listas de los individuos. Esta lista nos servirá más tarde para marcar que nodos no tienen comunidad asignada hasta el momento. Según recorramos las posiciones de los individuos, iremos sobrescribiendo estos -1 con el número de comunidad que le corresponda, el cual será un nuevo índice de comunidad si los nodos relacionados aún tienen un -1 o, en caso contrario, solo actualizará el índice que aun tenga un -1 en nuestra lista.

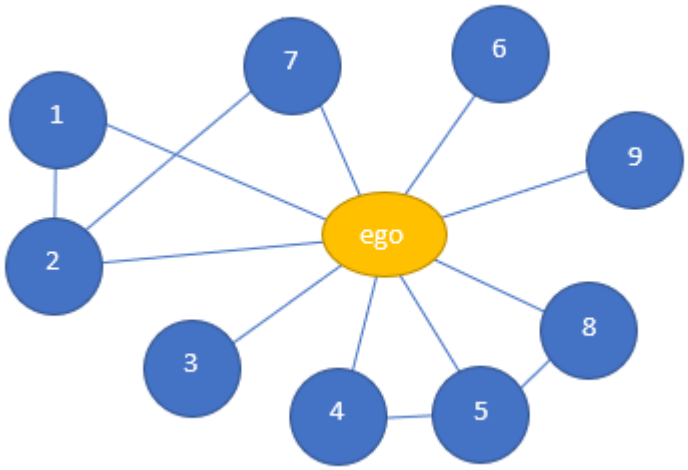

Figura 4-2: Grafo de prueba

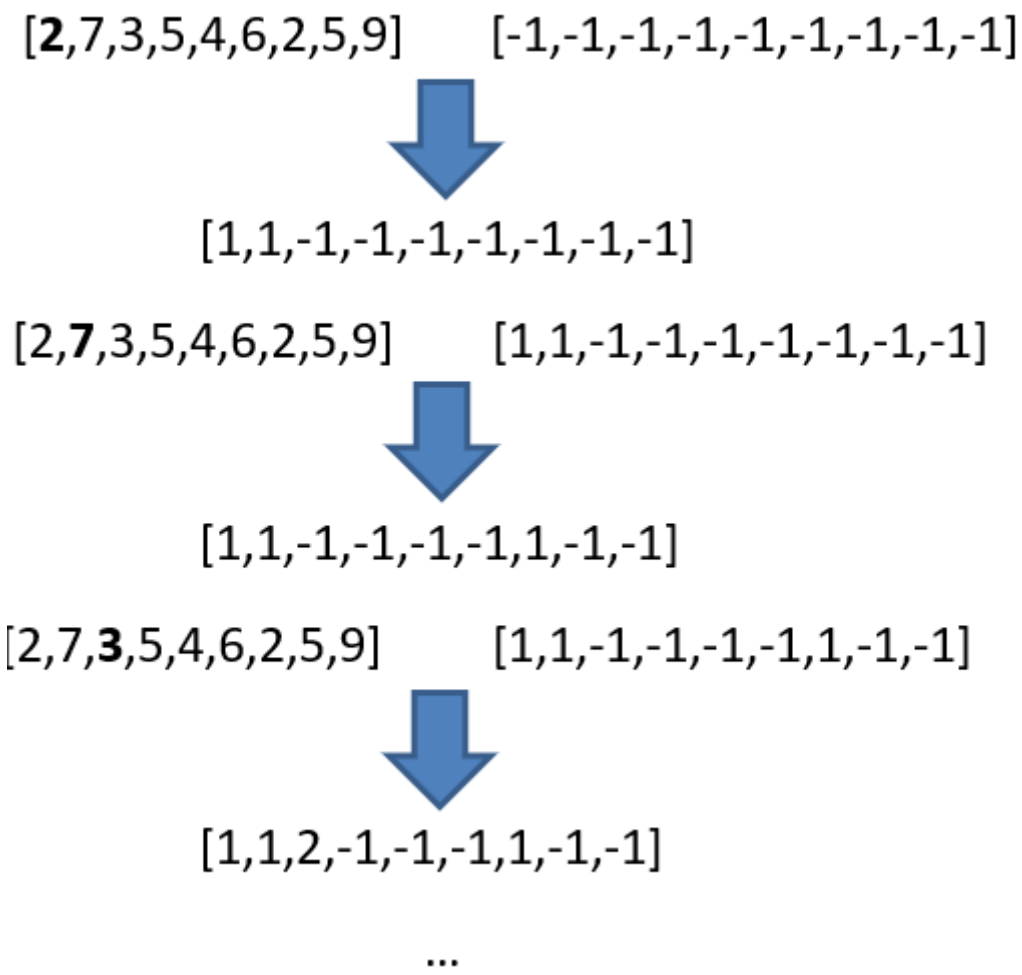

Figura 4-3: Ejemplo de localización de comunidades

Para aclarar la forma en la encontramos estas comunidades, vamos a fijarnos en la *Figura 4-3*, y en cómo se va actualizando nuestra lista de índices con un individuo correspondiente al mismo grafo de prueba *Figura 4-2* que hemos utilizado con anterioridad.

A la izquierda vemos la lista del individuo que se está analizando, y resaltado en negrita, el nodo al que se está accediendo. A la derecha, el estado actual de nuestra lista de comunidades encontradas, inicialmente a -1.

Leemos la primera posición, indicándonos que el nodo “1” tiene relación con el nodo “2”, por lo que nuestra lista de índices se actualiza colocando un 1 tanto en la primera como en la segunda posición. Accedemos a la segunda posición, donde nos encontramos que el nodo “2” tiene una relación con el “7”, como el nodo 2 ya tiene un valor de comunidad asignado, será la posición del nodo “7” la que se actualice, tomando el valor de la

comunidad del nodo con el que tenía relación, 1 en este caso. Para terminar, vemos como el nodo "3" se hace referencia a sí mismo, puesto que no dispone de vecinos en nuestro grafo, por lo que se asignará el nuevo índice de comunidad 2 en la tercera posición de nuestra lista de comunidades. Continuaremos así hasta que todas las posiciones de nuestra lista de comunidades tengan una comunidad asignada.

Al finalizar, tendremos los datos de nuestra población listos para poder obtener su valor de fitness.

4.2.3 Fitness.

Para determinar el factor de supervivencia de cada individuo, se ha dividido la función fitness en 3 partes: cálculo de la densidad inter-cluster y densidad intra-cluster y suma del valor de todas las densidades.

Para comenzar los cálculos se llamará a esta última, la cual se encargará de llamar y de obtener el valor de ambas densidades por cada comunidad que haya detectado un individuo.

Recordemos que, al llamar a nuestra función fitness, el valor de nuestros individuos ha pasado de contener el nombre de los nodos con los que se tenía un enlace, a tener una lista de posibles comunidades encontradas en el grafo, y será nuestra función fitness, la que determine la validez de las comunidades encontradas.

Por cada comunidad que porte un individuo, se llamara una vez a las funciones de cálculo de densidad inter-cluster e intra-cluster. Estas funciones realizarán los cálculos detalladas en las fórmulas *Figura 3-4* y *Figura 3-5* en el apartado de diseño. En el caso de la densidad intra-cluster es más sencillo, ya que devolveremos el resultado de la operación: len(subgraph.edges) / (0.5* len(subgraph.nodes) * (len(subgraph.nodes) - 1)). Donde subgraph es la comunidad encontrada por el individuo, edges aristas y nodes nodos.

Para el caso de la densidad inter-cluster, necesitaremos calcular la cantidad de conexiones que tienen los nodos de la comunidad con nodos fuera de ella (inter-cluster edges of C), para posteriormente poder aplicarla a la fórmula de retorno que será:

external_links / (len(community) * (len(self.network.nodes) - len(community))). Donde external_links será la cantidad de conexiones anteriormente mencionada, community los nodos que forman la comunidad y network toda la red ego.

Despues de cada llamada a estas funciones, se realizará su diferencia y se almacenará el valor en una variable acumuladora, donde se sumarán todos los resultados obtenidos por cada comunidad que porte un individuo.
result = result + (self.intracluster(community) - self.intercluster(community))

El resultado de esta suma será el fitness que finalmente se le asignará.

4.2.4 Cruce.

La función de cruce es la encargada de hacer evolucionar a la población, nos devolverá una lista de nuevos individuos con los que obtener comunidades en la siguiente generación. Como mencionamos en el apartado de elitismo 3.3.6, los mejores individuos no se modificarán, y se añadirán después de haberse realizado todos los cruces, por lo que la lista resultante de individuos será de un tamaño igual al del total de individuos – el porcentaje de elitismo declarado en él constructor.

Hasta completar ese número de nuevos individuos, se dará un valor aleatorio entre 2 y el número de individuos máximo a una variable *k,* usando de nuevo la librería random de Python. Este será el número de individuos que participarán en nuestro "torneo" de cruce.

Seleccionaremos esos *k* individuos participantes, otra vez, de manera aleatoria entre toda la población, para finalmente, quedarnos con los dos individuos con mejor fitness. Este proceso se llevará a cabo ordenando de mayor a menor la lista de *k* individuos y seleccionando los dos primeros. Por último, generaremos una máscara de 1's y 0's, de nuevo aleatoria, con el mismo tamaño que nuestros individuos, y en función del valor de cada índice, decidiremos escoger el valor del nodo del primer individuo o del segundo, tal y como se mencionó en el correspondiente apartado de diseño.

4.2.5 Mutación.

Para realizar la mutación, hemos creado una función donde se llevará a cabo todo el proceso. En ella, recorreremos todos los individuos ya cruzados de nuestra población, y generaremos un número aleatorio decimal entre 0 y 1 por cada posición en la lista del individuo, utilizando la ya mencionada librería random de Python. Si el número indicado para el porcentaje de mutación en el constructor del algoritmo, es menor o igual al obtenido aleatoriamente, el valor de esa posición en la lista mutará, si no, se mantendrá como estaba. La mutación se realizará como la indicada en el apartado de inicialización, ya que se tomarán todos los valores a los que puede optar esa posición y se cambiará por uno al azar.

5 Integración, pruebas y resultados

5.1 Tipos de pruebas realizadas.

Para poder determinar si la implementación del algoritmo llevado a cabo funciona según lo esperado, antes de probar a aplicarlo sobre las grandes bases de datos de Facebook de las que disponemos, se aplicará sobre unos pocos datos introducidos manualmente, con los que poder comprobar si el grafo esperado se crea correctamente y si las comunidades encontradas se corresponden con las dibujadas en el mismo.

Como se explicó en el anterior apartado de diseño, el algoritmo implementado requiere de varios parámetros de entrada con los que regular el funcionamiento de la mutación, elitismo, y delta de parada. Durante estas pruebas a menor escala, estos parámetros recibirán unos porcentajes altos con el fin de forzar su funcionamiento, y mediante impresiones por pantalla, comprobar que los valores de mutación a los que se acceden son los correctos, que el porcentaje de elitismo respecto a la población total funciona y que la condición de parada de nuestro delta se cumple correctamente. El cruce entre individuos se comprobará de la misma forma, viendo si los *k* individuos que participan en el torneo se seleccionan correctamente y que los valores de cada padre se escojan en función a la máscara generada aleatoriamente. Finalmente se comprueba que la formalización de nuestros datos a la de índices de comunidades estén bien indicadas y encontradas.

Como es natural, durante estas pruebas se encontró algún fallo en la implementación del algoritmo. La manera en la que habíamos determinado la formalización de nuestros datos no respetaba si un índice ya tenía una comunidad asignada, y si otro índice sin inicializar le hacía referencia, sobrescribía el valor que ya tenía guardado, generando más comunidades y de menor tamaño de lo que debería. Se cambió como se guardaban los individuos de diccionarios inicialmente, a listas para facilitar el uso de la función fitness de la que se dispone.

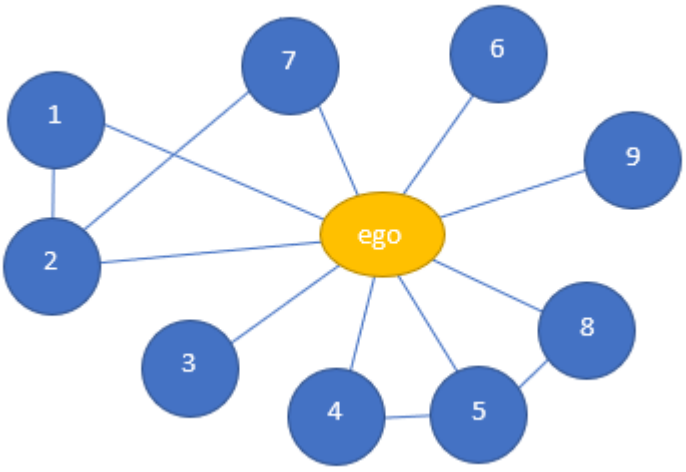

Figura 5-1: Grafo de prueba

Utilizando el mismo grafo usado durante las explicaciones, se puede observar a simple vista que nuestra lista de nodos será [1, 2, 3, 4, 5, 6, 7, 8, 9] (por orden de lectura) y las comunidades que encontramos a simple vista son [[1, 2, 7], [3], [4, 5, 8], [6], [9]].

```
ipdb> [[[1, 2, 7], [3], [4, 5, 8], [6], [9]], 0.26666666666666666]
```

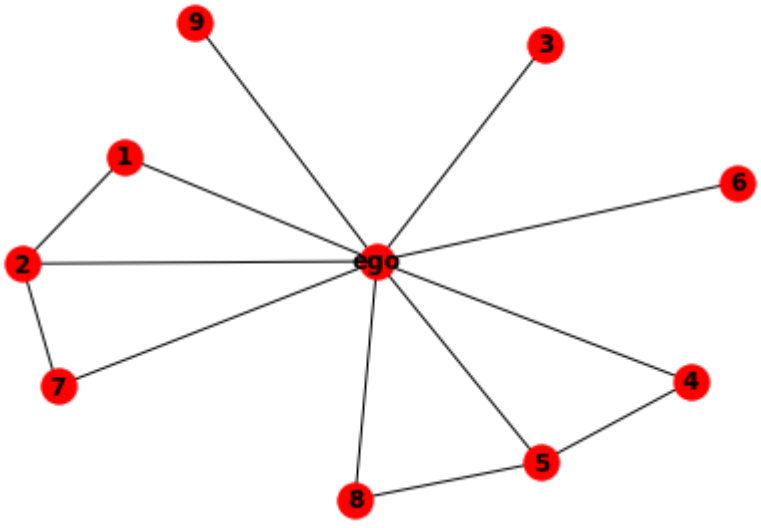

Figura 5-2: Resultado salida de prueba

Aplicamos nuestro algoritmo genético sobre estos datos y obtenemos la salida mostrada en *Figura 5-2.*

Como podemos observar, al ser unas comunidades muy bien definidas y con pocos datos, nuestro algoritmo encuentra correctamente las comunidades esperadas, y utilizando la función que tiene Networkx para dibujar el grafo creado, obtenemos la representación gráfica deseada de los datos introducidos. Seguido de la lista de comunidades encontradas, se puede observar el fitness obtenido por el algoritmo genético (0.26666666666666666), que aun siendo un tanto bajo, es comprensible debido a la baja densidad de nodos en las comunidades, y entre ellas, por la escasa cantidad de nodos en total.

Una vez obtenido este resultado y sabiendo que el algoritmo genético implementado funciona según lo esperado, se procede a la aplicación del mismo sobre los datos reales que se dispone sobre la red social Facebook [12].

5.2 Resultados obtenidos y discusión.

5.2.1 Resultados

A continuación, se mostrará una tabla con los diferentes valores obtenidos tras la aplicación del algoritmo implementado sobre varios de los dataset probados. Estos valores se obtuvieron sobre el mismo dataset, pero modificando los parámetros de entrada para observar las posibles variaciones en la salida que se pudieran dar.

Ego 0 (ficheros 0.edges, 0.feat) con 347 nodos

N.º de individuos	N.º de generaciones realizadas	% de mutación	% de elitismo	Delta	Mejor fitness medio
25	40	0,1%	5%	0,0001	0,777872135
50	44	0,1%	5%	0,0001	0,816018187
100	41	0,1%	5%	0,0001	0,827105456
500	43	0,1%	5%	0,0001	0,852008969
1000	40	0,1%	5%	0,0001	0,873405805

2000	30	0,1%	5%	0,0001	0,871665527

Tabla 1: Ejecuciones del algoritmo sobre ego 0

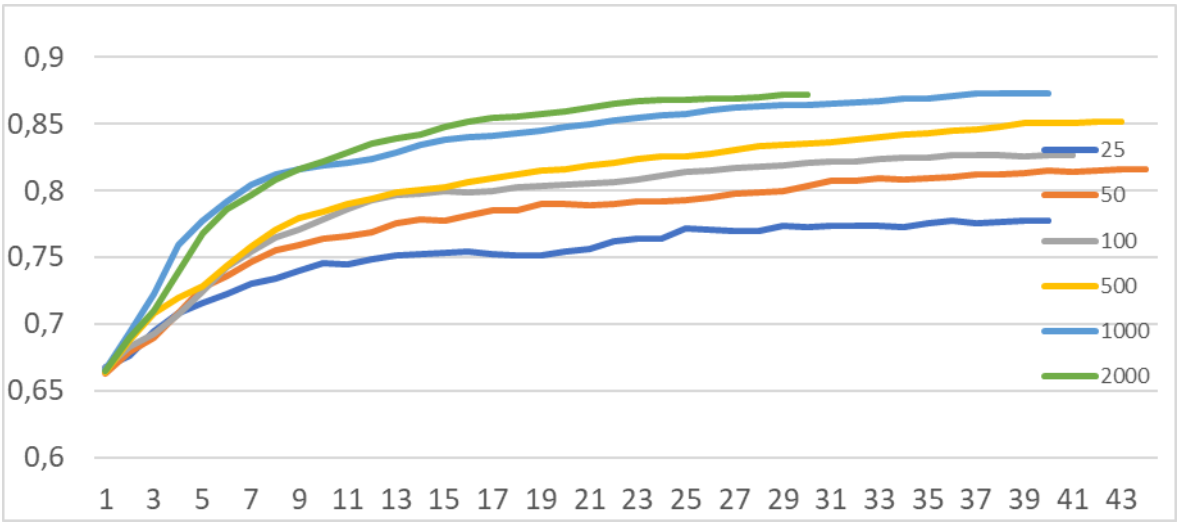

Figura 5-3: Evolución fitness medio en ego 0

Ego 414 (ficheros 414.edges, 414.feat) con 159 nodos

N.º de individuos	N.º de generaciones realizadas	% de mutación	% de elitismo	Delta	Mejor fitness medio
25	27	0,1%	5%	0,0001	0,816773005
50	21	0,1%	5%	0,0001	0,83363216
100	18	0,1%	5%	0,0001	0,856482306
500	21	0,1%	5%	0,0001	0,866039765
1000	22	0,1%	5%	0,0001	0,873878406
2000	25	0,1%	5%	0,0001	0,882715524

Tabla 2: Ejecuciones del algoritmo sobre ego 414

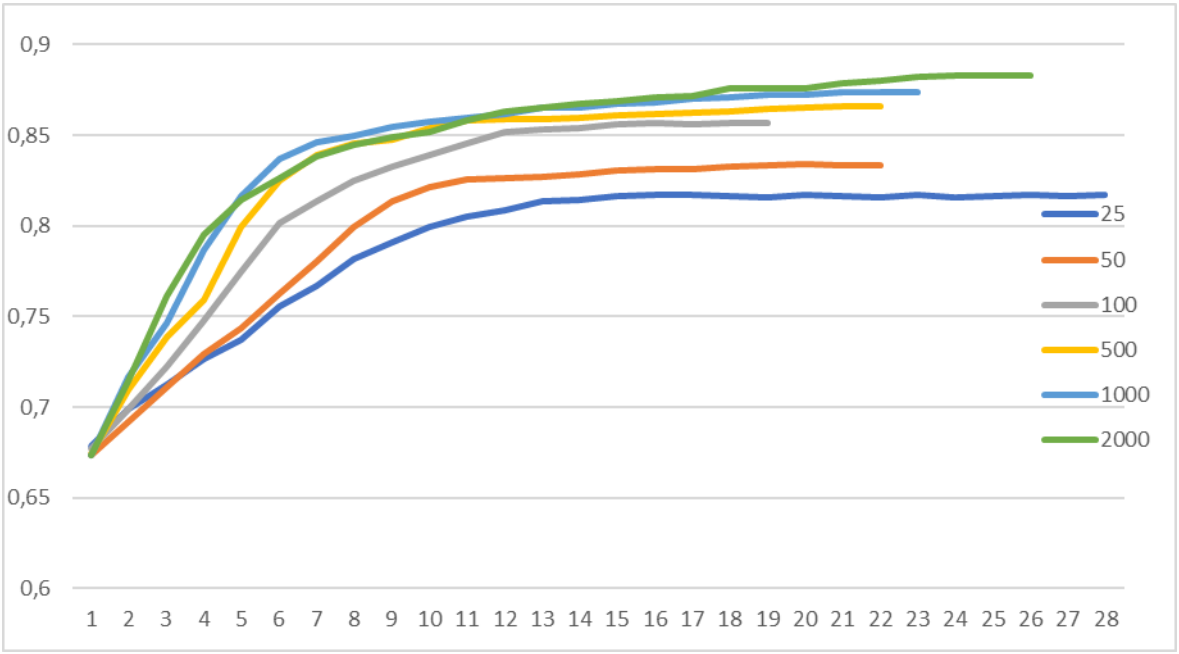

Figura 5-4: Evolución fitness medio en ego 414

Ego 0 (ficheros 0.edges, 0.feat) variación de elitismo

N.º de individuos	N.º de generaciones realizadas	% de mutación	% de elitismo	Delta	Mejor fitness medio
500	41	0,1%	0%	0,0001	0,859590923
500	44	0,1%	5%	0,0001	0,864534389
500	37	0,1%	15%	0,0001	0,85979792
500	34	0,1%	30%	0,0001	0,845490342

Tabla 3: Ejecuciones del algoritmo sobre ego 0 con variación de elitismo

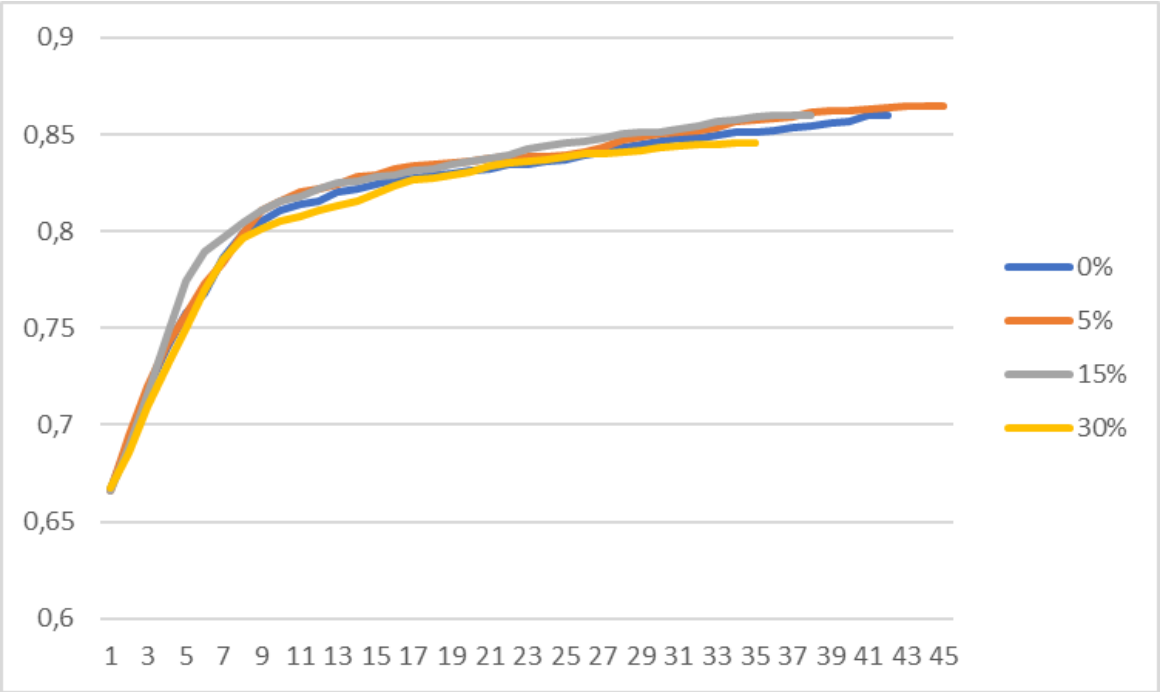

Figura 5-5: Evolución fitness medio en ego 0 con variación de elitismo

Ego 414 (ficheros 414.edges, 414.feat) variación de mutación

N.º de individuos	N.º de generaciones realizadas	% de mutación	% de elitismo	Delta	Mejor fitness medio
500	10	0%	5%	0,0001	0,830787607
500	28	0,1%	5%	0,0001	0,873746105
500	20	0,5%	5%	0,0001	0,864333539
500	18	1%	5%	0,0001	0,865258885
500	16	5%	5%	0,0001	0,808084169
500	50	10%	5%	0,0001	0,756445695

Tabla 4: Ejecuciones del algoritmo sobre ego 414 con variación de mutación

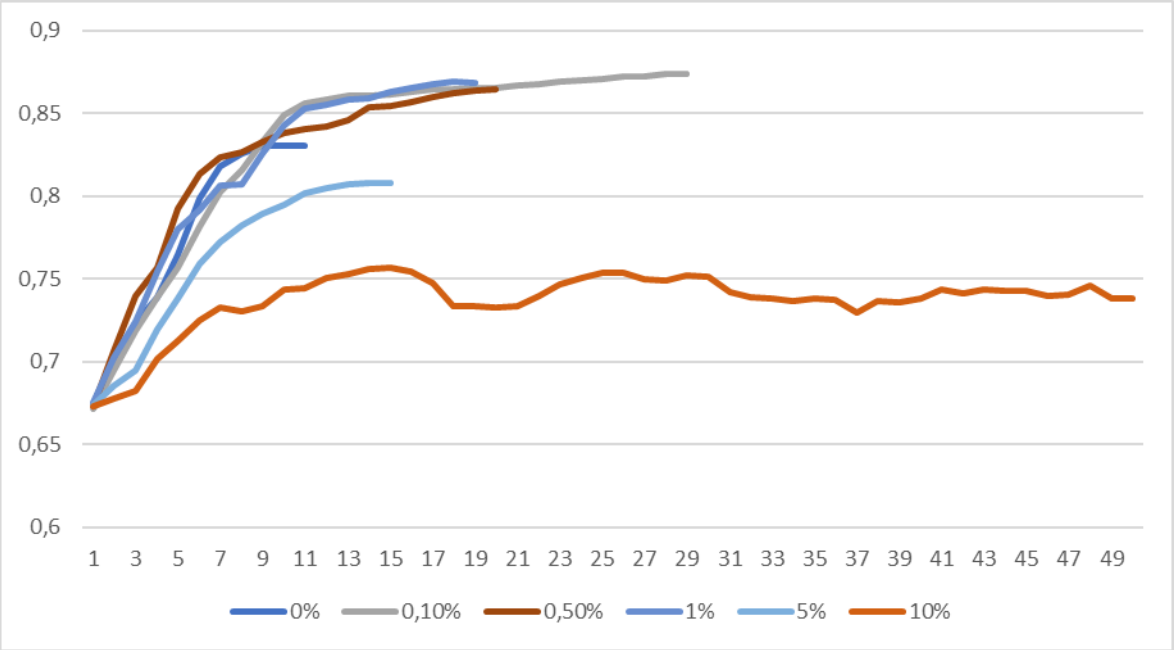

Figura 5-6: Evolución fitness medio en ego 414 con variación de mutación

5.2.2 Discusión sobre los resultados obtenidos

Si nos paramos a observar los resultados obtenidos, podemos ver como el fitness medio de la población mejora si aumentamos el número de individuos que la componen. Esto parece lógico, puesto que cuanto mayor sea el número de individuos que compongan nuestra población, mayor será la diversidad entre ellos y se encontrará un mayor número de soluciones posibles, con el correspondiente aumento en el número de posibilidades que surgen al cruzarlos, por lo que podemos determinar sin ninguna duda en la importancia de que el número de individuos sea lo más grande posible. Como contra parte, tenemos que en el paso de 1000 individuos a 2000, el fitness medio no solo no mejora, si no que parece empeorar un poco, y probando con números más grandes de individuos el resultado parece estancarse en ese 0,87. Con este dato, podemos deducir que aumentar el número de individuos no nos aportará ningún fitness mucho mejor del que ya obtenemos con estos 1000, y solo hará que nuestro algoritmo aumente su tiempo de ejecución considerablemente.

Otro dato destacable es el delta tan bajo usado para las pruebas (0,0001), esto es debido a la baja mejora que se puede apreciar en las gráficas de la evolución del fitness, empezando generalmente en un 0,6 y mejorando en torno a 40 generaciones hasta el 0,87

anteriormente indicado, suponiendo una mejora media por generación del 0,00675 en el fitness. Al inicializar el delta en un valor mayor, el algoritmo paraba entre las 5 y 10 generaciones, dejándonos un máximo que, como hemos visto, es mejorable si de deja más tiempo. De esto deducimos que, llegado a un punto, el algoritmo no tiene mucha flexibilidad a la hora de generar más diversidad en los individuos, y las mejoras aumentan muy lentamente hasta llegar a ese tope anteriormente señalado. Es probable que, el culpable de esta poca diversidad sea la forma en la que el algoritmo produce las mutaciones, ya que el valor de un nodo solo puede mutar al de los adyacentes al mismo, limitando la creación de más diversidad. El número de nodos en el grafo también influye en el número de generaciones que se crean antes de llegar al delta, como se puede apreciar en *Tabla 2,* donde el dataset consta de menos nodos y el número de generaciones es también menor al de *Tabla 1*.

Respecto a los datos obtenidos sobre los porcentajes de elitismo, decidimos usar el 5% de los individuos totales tras realizar pruebas con varios porcentajes y ver que era el que mejores datos nos devolvía. No obstante, la diferencia de los resultados con pequeñas variaciones de elitismo es bastante baja y no se empezará a notar un peor rendimiento hasta que se aplique el 40% de elitismo, reduciendo drásticamente el número de individuos que pueden mutar y cruzar, lo que a su vez reducirá la diversidad de las posibles soluciones.

Por último, es importante destacar la elección del porcentaje de nodos que mutan durante la ejecución. Si bien vemos, que negando la posibilidad de que los nodos muten el algoritmo seguirá encontrando una solución asequible, esta será con pocas iteraciones e inferior a la de otros porcentajes, cuando permitimos que los nodos muten, añadimos la capacidad al algoritmo de encontrar nuevas y mejores soluciones de las que se obtendrían solo cruzando los individuos, pero como en la naturaleza, un exceso en el porcentaje de mutación puede llevar a que el algoritmo no logre evolucionar hacia la dirección correcta y se quede fluctuando sobre un valor (como en el caso del 10%). A la vista de las pruebas realizadas, es importante que se aplique un porcentaje muy bajo de mutación, siendo 0,1% el elegido para realizar el resto de las pruebas ya que fue el que mejores resultados nos ofreció.

Al compararse las diferentes comunidades obtenidas con las del dataset, se observó como en la mayoría de los casos las comunidades encontradas eran de menor tamaño que en las

del fichero circles.*x* correspondiente a cada ego, generándose más comunidades y de menor tamaño en media. Aunque algunos nodos coincidían en la misma comunidad en ambas salidas, también se descubrió que el fichero circles añadía en algunos casos al mismo nodo en varias comunidades diferentes, echo que en nuestro algoritmo no se puede dar.

No se realizó una comparación más exhaustiva con un Omega Index por falta de tiempo durante la realización de este Trabajo de Fin de Grado.

Aunque los resultados obtenidos de aplicar el algoritmo implementado sobre los datos de Facebook diste de los que el fichero circles de cada ego nos aporta, no podemos deducir que el resultado obtenido sea incorrecto o poco fiable, puesto que las comunidades detalladas en el fichero circles pueden estar formadas por otros aspectos o características más allá de la conexión entre usuarios, tales como el lugar de residencia, estudios, gustos personales… además de la posibilidad de asignar el mismo nodo a varias comunidades. Aun así, que ciertos nodos se les ubique en ambas salidas en la misma comunidad, es algo muy positivo a tener en cuenta en el resultado del algoritmo.

Por desgracia, la función que dispone Networkx para pintar los grafos resultantes, no nos permite crear imágenes más grandes de las que hemos mostrado anteriormente, haciendo que los grafos con una gran cantidad de nodos como los que estamos creando, sean muy difíciles de ver correctamente como en el caso de *Figura 5-7.*

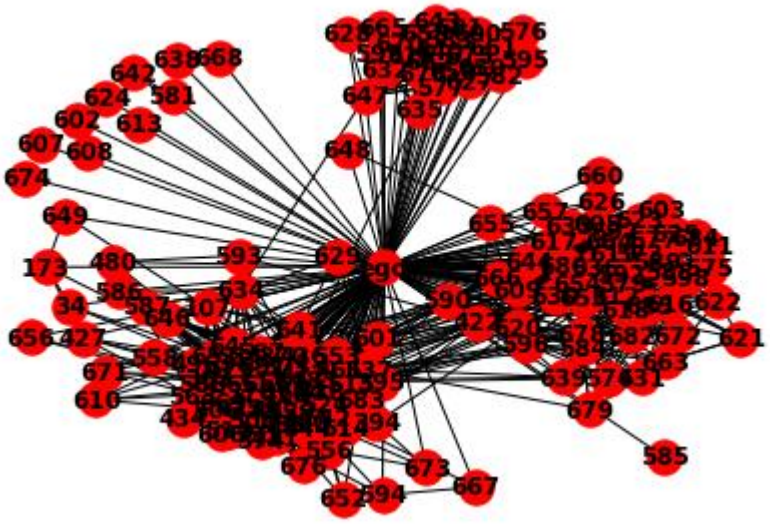

Figura 5-7: Grafo resultado de aplicar sobre ego 414

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Con este Trabajo de Fin de Grado se ha tratado de demostrar que es posible la aplicación de algoritmos genéticos para resolver problemas como la detección de comunidades en las diferentes redes sociales, representadas a su vez en forma de redes ego.

La detección de comunidades ha ganado un gran interés debido a la importancia y crecimiento que las redes sociales han tenido en los últimos años, convirtiéndose en un referente en cuanto a comunicación e información. Utilizamos estas redes para prácticamente cualquier cosa, transmitiendo enormes cantidades de información en forma de opiniones, gustos o incluso siguiendo y creando relaciones entre otros usuarios de la red. Por todo esto, el análisis de estas redes se ha convertido en un campo donde los investigadores pueden obtener gran cantidad y diversidad de datos, desde el estudio de las ciencias sociales hasta la informática.

Al utilizar grafos para representar las estructuras formadas por estas redes, podremos entender más fácilmente que estructuras forman sus enlaces, como se han creado estas conexiones entre los individuos, y lo que es más interesante, llegar predecir el comportamiento de los usuarios. Si encontramos una forma de detectar las comunidades que se crean dentro de las redes sociales, conseguiremos establecer similitudes en las características de los usuarios que componen estas comunidades y viceversa.

Las pruebas realizadas en este Trabajo de Fin de Grado, tanto en los primeros grafos de prueba para comprobar su funcionamiento como con los dataset de Facebook, han demostrado que el uso de un algoritmo genético tiene potencial para resolver el problema propuesto en esta memoria, y que, con los ajustes necesarios en los parámetros de entrada, se pueden obtener un fitness optimo con una buena elección de comunidades.

6.2 Trabajo futuro

Si bien el algoritmo implementado cumple con los objetivos que se presentaron en este Trabajo de Fin de Grado, tiene ciertos problemas a la hora de enfrentarse a grafos con un número de nodos muy grande, ya que el tiempo de ejecución en estos casos se dispara cuando se realizan pruebas con un número de individuos de 1000 o superior.

Para futuros trabajos, se deberá llevar a cabo la optimización de la forma en la que el algoritmo relaciona los nodos con las comunidades, puesto que recorrer cada individuo comprobando a que comunidad pertenece cada nodo es muy costoso. De esta forma, mejoraríamos notablemente el funcionamiento del algoritmo, ya que se podrían realizar más pruebas en menor tiempo.

También se pueden llevar a cabo comparaciones más exhaustivas con un Omega Index sobre las soluciones que estos datasets nos aportan, ya que, con la falta de tiempo a la hora de realizar estas primeras pruebas, no se pudieron realizar.

Realizar comparaciones con otros fitness u otros métodos de búsqueda de comunidades que aporten soluciones parecidas, para así tener una mejor visión de hasta qué punto los resultados son buenos o se pueden mejorar.

Referencias

[1] C. Darwin (1859). On the Origin of Species by Means of Natural Selection, Murra y, London.

[2] D.E. Goldberg (1989). Genetic Algorithms in Search, Optimization and Machine Learning , Addison-Wesley , Reading, MA.

[3] J. Holland (1975). Adaptation in Natur al and Articial Systems , Universit y of Michigan Press, Ann Arb or.

[4] L. Davis (ed.) (1991). Handbook of Genetic Algorithms , Van Nostrand Reinhold, New York.

[5] Z. Michalewicz (1992). Genetic Algorithms + Data Structures = Evolution Programs , Springer-Verlag, Berlin Heidelb erg.

[6] C. Reeves (1993). Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scien-tic Publications.

[7] Qing Cai, Maoguo Gong, Lijia Ma, Shasha Ruan, Fuyan Yuan, Licheng Jiao, "Greedy discrete particle swarm optimization for large-scale social network clustering", International Research Center for Inteligent Perception and Computation, Xidian University, China.

[8] Sato,S., K. Otori, A. Takizawa, H. Sakai, Y. Ando y H. Kawamura. (2002). "Applying genetic algorithms to the optimum designo f a concert hall" *journal of sound and vibration*, vol.258, no.3, p. 517-526.

[9] Obatashi, Shigeru, Daisuke Sasaki, Yukihiro Takeguchi, y Naoki Hirose. (julio de 2000). "Multiobjective evolutionary computation for supersonic wing-shape optimization." *IEEE transactions on Evolutionary Computation*, vol.4, no.2, p.182-187.

[10] Mahfound, Sam y Ganesh MMani. (1996). "Financial forecasting using genetic algorithms." *Applied Artificial Intelligence*, vol.10 no.6, p.543-565.

[11] Equipo docente de la asignatura "Fundamentos de Aprendizaje Automático" de la UAM "Algoritmos Genéticos", EPS-UAM

[12] Jure Leskovec, Snap Facebook dataset, [online] Available at: https://snap.stanford.edu/data/egonets-Facebook.html.

[13] Wikipedia "Teoría de grafos", [online] Available at : https://es.wikipedia.org/wiki/Teoría_de_grafos#Historia.

[14] Documentación de la librería Networkx de Python, [online] Available at: https://networkx.github.io/documentation/stable/tutorial.html.